# Design and Analysis of Algorithms Assignment 1

Harrison Lee, Alex Zhao

January 25, 2019

**Question 1.** *(3.9 "Bottleneck" Nodes in a Graph)*

Claim: Given two nodes, $s$ and $t$, in $n$-node undirected graph $G = (V, E)$ with a distance greater than $n/2$, there exists some node $v$ not equal to either $s$ or $t$ that, when deleted, destroys all paths from $s$ to $t$.

*Proof.* A path of distance greater than $n/2$ takes $n/2 + 1$ nodes at least. Excluding $s$ and $t$ this is $(n-2)/2 + 1 = n/2$ nodes. Let's call this path "path $A$".

For node $v$ to be deletable without destroying the other path, path $B$, $v$ cannot be in $B$.

$B$ must also have a distance greater than $n/2$ to maintain $s$ and $t$s' distance.

$B$ cannot share nodes with $A$ other than $s$ and $t$ and requires $n/2$ nodes unique from $A$.

There are only $n - 2$ non-$s$ or non-$t$ nodes, but $A$ and $B$ require a total of $n$ unique nodes, so $B$ cannot exist.

□

**Algorithm 1.** *Find node v*

Begin with a Depth-First-Search, as written in the textbook. Use it to find the shortest path from $s$ to $t$.

Mark all nodes discovered in that shortest path as "Used", numbering them by their distance to $t$.

Repeat Depth-First-Search, starting from $s$, but do not traverse past "Used" nodes. Instead, mark them as "Re-Found". Save whichever "Re-Found" node that is closest to $t$.

Once Depth-First-Search fails and cannot continue, return the saved "Re-Found" node as $v$.

*Proof.* Prove this algorithm is $O(n + m)$:

Depth-First-Search is known to be $O(n + m)$, as noted in the textbook. Each edge and node is traversed at most once.

This algorithm conducts Depth-First-Search twice.

This algorithm is $O(2n + 2m)$, which is close enough to $O(n + m)$ for our purposes.

□

**Question 2.** *(4.6)*

Given: Contestants $c_i$ with projected swimming times $s_i$, biking times $b_i$, and running times $r_i$. Only one contestant may swim at one time and each contestant must finish swimming before they may begin running and biking.

Find: An order for the contestants to start that minimizes the time it takes for all the contestants to complete the triathlon.

**Algorithm 2.** *Contestants start in the order of descending $b_i + r_i$. This is $O(n \cdot log(n))$ (sorting)*

*Proof.* Suppose there is an optimal schedule where contestants are not ordered by descending $b_i + r_i$. That means there exists at least one "inversion" where $c_i$ starts before $c_j$ but $b_i + r_i \leq b_j + r_j$. Swapping this inversion will not increase the time it takes for all contestants to complete the triathlon.

After the swap, $c_j$ will finish earlier because they enter and leave the pool earlier, meaning they do not increase the overall completion time.

After the swap, $c_i$ will finish swimming at the same time $c_j$ would have finished before the swap (this is because total swimming time of all the contestants up to and including $c_j$ before the swap is still equal to the total swimming time of all the contestants up to and including $c_i$ after the swap. Because $b_i + r_i \leq b_j + r_j$, $c_i$ will not finish running later than $c_j$ would have before the swap.

Therefore, swapping an inversion does not increase the time increase the time it takes for all contestants to complete the triathlon, meaning we can continue to swap these inversions until there are no inversions remaining and the ordering of contestants in the optimal solution is the same as the one given by our algorithm.

□